Sponsored by

Defense Advanced Research Projects Agency (DoD)

Defense Small Business Innovation Research Program

Highly Parallel Iterative Methods for Massively Parallel Multiprocessors

ARPA Order No. _____

Issued by U.S. Army Missile Command Under

Contract No. DAAH01-88-C-0409

Approved for public release; distribution unlimited.

Scientific Computing Associates, Inc.
246 Church Street, Suite 307
New Haven, CT 06510

Effective Date of Contract: 01 September 1988
Contract Expiration Date: 28 February 1989
Reporting Period: 01 September 1988 to 18 February 1989
Principal Investigator: Dr. David E. Foulser
Telephone: (203) 777-7442
Short Title: Iterative Methods for the Connection Machine

DISCLAIMER

The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the Defense Advanced Research Projects Agency or the U.S. Government.

DESTRUCTION NOTICE

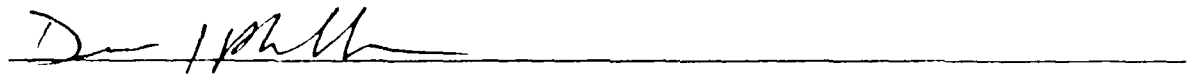
Follow the procedures delineated in paragraph 19 of Industrial Security Manual, DoD 5220.22-M.

CERTIFICATION OF TECHNCIAL DATA CONFORMITY

The Contractor, Scientific Computing Associates, Inc., hereby certifies that, to the best of its knowledge and belief, the technical data delivered herewith under Contract No. DAAH01-88-C-0409 is complete, accurate, and complies with all requirements of the contract.

Date: February 28, 1989

Name and Title of Certifying Official:



Highly Parallel Iterative Methods for Massively Parallel Multiprocessors

Final Report on Phase I Research

1. Abstract

In the Phase I work presented here, we developed computer programs for the Connection Machine (a CM-2) to compute sparse matrix-vector multiplies, sparse triangular solves, and inner-products. These subproblems constitute the iterative portion of Krylov space methods that use incompletely factored preconditioning matrices to solve very large sparse linear systems. The numerical solution of such systems is very often the compute-intensive kernel of the kind of large scale engineering and scientific computations that are commonly done on supercomputers. Moreover, Krylov space methods which are preconditioned by appropriate incompletely factored matrices are generally recognized as the algorithms of choice in terms of minimizing the number of floating point operations required for solving such problems.

We performed a variety of benchmarks on large three-dimensional model problems over cube-shaped domains discretized with a seven point template. The highest computational rate we were able to achieve for the sparse triangular solve was 13.1 MFlops on 4K processors; this would correspond to a timing of 210 MFlops on an appropriately scaled problem on a 64K processor machine. The highest computational speed we achieved for a matrix vector multiply was 64.2 MFlops, this would correspond to to a speed of 1027.0 MFlops on a 64K processor machine. We compared the computational speed obtained from the CM-2 with that obtained from equivalent highly-optimized code on a single head of a Cray X/MP. For all three computational kernels the projected speeds on a full 64K processor machine exceed the Cray X/MP speeds by a factor of at least 5 to 6. Thus for regularly-structured problems, the CM-2 achieves impressive computational speeds. When one takes into account the price differential between these two-systems, one realizes that the price-performance advantage for the CM-2 over a single-headed Cray X/MP is truly outstanding.

The timings on the CM-2 were highly dependent upon the use of our very specialized programs. These programs mapped the problem domain onto the processor topology very carefully and used the optimized local NEWS communications network. We explored the consequences of relaxing these restrictions in a variety of ways. Performance degraded very significantly as we abandoned the optimized NEWS communications network and abandoned the careful mapping of a problem domain to the CM-2 processor topology. In some cases we noted 20 to 75 fold degradations in performance as these constraints were relaxed.

2. Overview

There are at least two-critical components required to obtain extremely fast methods for solving linear systems. One is the use of efficient and robust numerical algorithms, and the other is the employment of effective techniques for delivering a large amount of computing power. These requirements can conflict with one another in a variety of ways. Many modern methods of solving reasonably general classes of linear systems involve a degree of implicitness; this implicitness can limit the amount of available concurrency.

When Krylov space linear solvers are used, the choice of preconditioner can play a major role in determining the operation count of the resulting algorithm [7, 10, 11, 12]. Unfortunately, some of the most powerful preconditioners are obtained by using incompletely factored matrices. To apply these preconditioners, we must repeatedly solve sparse triangular systems. The efficiency with which such solutions could be carried out was characterized by Saad and Schuitz [14]. Sparse triangular systems arising from a range of problems have been solved efficiently on a number of shared memory architectures [1], [3], [4], [9]. Because data dependencies limit the concurrency available from a sparse triangular solve, it has not been clear that triangular solves could be employed usefully in programs written for massively parallel architectures. One goal of our study was to determine whether one could realistically hope to take advantage of incompletely factored matrices as preconditioners for solving sparse linear systems on massively parallel architectures such as the CM-2.

Several other researchers have addressed the relative utility of various forms of preconditioning on the CM-2. Both [8] and [13] concluded that it was not worthwhile to precondition using incompletely factored matrices when solving linear systems arising from two-dimensional partial differential equations on massively parallel machines such as the CM-2. We concentrated our efforts on a simple three-dimensional model problem because of the greater parallelism found in the sparse triangular solve.

Preconditioning based on polynomials often has been suggested as a alternate preconditioning that is highly parallel. While such preconditioners may achieve high computation rates, the overall picture is not clear since they often require more iterations. One goal of our study was to consider whether the faster computation rate of a polynomial preconditioner on a massively parallel machine such as the CM-2 would overcome the greater number of iterations required to converge to a solution.

In the work described in this Phase I final report, we have carefully examined a set of model problems and, under the appropriate circumstances, we are able to obtain surprisingly good performance on key computational kernels including the sparse triangular solves used for preconditioning in Krylov space solvers. To put our results in perspective, we compared our measured and projected computational speeds with results from experiments on a single head of a Cray X/MP. We concluded that for selected problems, a 64K processor CM-2 could outperform an analogously well-tuned program run on a single head of a Cray X/MP by at least a factor of 5 in performing the sparse triangular solves and the matrix vector multiplies required in the Krylov solver inner loop.

Efficient methods for solving partial differential equations frequently make use of non-uniform grids designed to put the most computational effort where the problem is hardest. An effect of this approach is that the algebraic linear (and non-linear) systems that must

A-1

eventually be solved are sparse and quite irregular in structure. Careful mapping of workload can be extremely important in obtaining adequate performance from multiprocessor architectures with strong memory hierarchies; mapping is typically straightforward in very regular problems with a known structure and is much more problematic for problems with unknown or irregular structures.

Another goal of our Phase I research was to quantify the degree to which performance on a machine such as the CM-2 depends on exploiting regularities in problem structure. We will present a number of timings on the CM-2 for sparse matrix vector multiplies, sparse triangular solves and inner products, which show that obtaining good performance on the CM-2 is critically dependent on the use of very well-tuned special-purpose computational kernels. We have described a number of experimental results arising from our investigations in [5] and [6].

In Section 3, we present what we regard as best case timings for the sparse matrix vector multiplies, sparse triangular solves, inner products and SAXPYs that can constitute the iterative portion of Krylov-based programs. The timings show that for large three-dimensional problems, good performance can be obtained from sparse triangular solves. In Section 4, we demonstrate that performance on the CM-2 is an extremely sensitive function of 1) problem mapping and 2) *a priori* knowledge of dependency patterns. We will show that this performance sensitivity shows up very strongly not only in the sparse triangular solves but also in the sparse matrix vector multiply.

3. Performance on a Regular Three-Dimensional Mesh

In this section we describe the results of experiments that give a best case estimate of the rate with which the CM-2 can carry out the portions of Krylov space linear equation solvers. We will first present timings from consecutive sweeps over a three-dimensional mesh along with timings for the corresponding inner products and SAXPYs. The performance measurements we obtain here characterize the performance that would arise from the iterative portions of linear solvers employing many simple preconditioners. We will then present timing results from a program that performs a sequence of linked matrix vector multiplies and triangular solves. We argue that the matrix vector multiply and triangular solve timings obtained from this benchmark are a fair measure of the timings that would be observed from these procedures were they integrated into an iterative loop of the appropriately preconditioned Krylov solver. As part of this test loop, we also measured the time required to perform inner products in a manner that conformed with data structures and the mapping used for the other two procedures.

The software provided with the CM-2 makes use of the concept of *virtual processors*; one can program the CM-2 so that it appears that there are a larger number of processors than actually exist. The CM-2 software assigns blocks of virtual processors to each real processor. This assignment of multiple virtual processors to each real processor tends to amortize the overhead of transmitting each instruction to the physical processors. In most of the problems we investigated in Phase I, increasing the ratio of virtual to real processors reduced overheads due to communication.

3.1. Mesh Sweeps

One can use a very large number of virtual processors in implementing a sparse matrix vector multiply. We examined the performance of a very specialized PARIS program (PARIS is the CM-2 assembly language) which was written for a three-dimensional problem on a cube with a seven-point operator. The program consisted of a sequence of sweeps over a three-dimensional mesh. The mesh was embedded into a cube of virtual processors with one mesh point assigned to a virtual processor. The cube of virtual processors used by the program has an edge size equal to a power of two. Subject to this constraint, the largest mesh we could embed was one with an edge size of 64. Each iteration of the 1000 carried out took an average of 49 milliseconds. This corresponds to a speed of 64.2 MFlops on the 4K processors. With an appropriately scaled problem, one might expect to obtain 1027.0 MFlops on a 64K processor machine (appropriately scaled means keeping the virtual processor ratio fixed). The results obtained from timings for meshes of varying sizes on 4K processors are shown in Table 1.

In Table 1 we also depict measurements obtained from SAXPYs and inner products over three-dimensional domains. All of these results were obtained by timing 100,000 consecutive iterations. Because SAXPYs do not require communication, we expect to obtain extremely high performance. For SAXPYs carried out over a cube of virtual processors with edge size 128, we obtained a speed on 4K processors of 235.0 MFlops, which would correspond to 3760 MFlops on a 64K processor machine. The efficiency with which SAXPYs are performed decreases when one uses fewer virtual processors. A cube with edge size 16 has one virtual processor for each actual processor. From Table 1 we see that the speed of this computation decreases to 131.0 MFlops — note that this reduction in speed

Grid Size edge	MVM MFlops	SAXPY MFlops	Inner Product MFlops
16	23.5	131.0	14.6
32	46.1	226.0	81.3
64	64.2	233.9	185.8
128	N.A.	235.0	220.9

Table 1: Three-Dimensional Embedding
Mesh Sweeps, Inner Products, SAXPYs
4K processors

must be unrelated to communication overhead. In Table 1, we also present timings for inner products carried out over a cube of virtual processors with varying edge size.

3.2. Performance from Iterative Loops with Triangular Solves

We next present timing results from a program that performs a sequence of linked matrix vector multiplies and triangular solves. Let M represent a matrix obtained from the uniform discretization of a cube with a seven point template and let L represent a lower triangular matrix with the same sparsity structure as M. We carried out the test calculation depicted in the program below.

```
do 100 times

  Mx = y
  Solve Lx = y
  z = inner-product(y,y)

end do
```

This program carried out the matrix vector multiply $Mx = y$ by sweeping over a three-dimensional mesh. We embedded the three-dimensional mesh into a two-dimensional gray-coded processor lattice. The sparse lower triangular system of equations $Lz = y$ was then solved by sweeping over another three-dimensional mesh, embedded in a conforming

fashion into the same two-dimensional processor lattice. The sweep used to solve the sparse triangular system was carried out in a manner that respected the dependencies of the problem. As part of the test loop, we also measured the time required to perform inner products in a manner that conformed with data structures and the mapping used for the other two procedures.

We now describe in more detail how the triangular solve was carried out. In a cube with n points along any edge, i, j, k between 1 and n are used to define the position of a point in the cube where i, j, k represent the cartesian coordinates of a point in the 3-D mesh. We can parallelize this three-dimensional triangular solve by concurrently solving, for each consecutive v, the plane of points satisfying the condition $i + j + k = v$ for $1 \leq v \leq 3n - 2$. Each processor in the lattice contained, for a given i and j, variables corresponding to values of k between 1 and n.

Table 2 depicts the timings obtained for various size domains for 4K processors, timings from 100 iterations were averaged. The timings obtained from the cube with edge sizes 128 and 64 corresponded to 188.9 and 104.9 MFlops respectively for the inner product, 52.6 and 27.1 MFlops respectively for the matrix vector product and 13.1 and 7.0 MFlops respectively for the triangular solve. From the above results we conclude that for an appropriately scaled problem on 64K processors, the fastest results we could obtain for the inner product, matrix vector multiply and triangular solve would be 3022 MFlops, 842 MFlops and 210 MFlops respectively.

Grid Edge Size	Inner Product (ms)	Matrix Vector (ms)	Triangular Solve (ms)	Total Time (ms)
16	1.5	29.0	51.9	81.3
32	2.8	56.7	106.8	162.4
48	3.8	78.4	169.1	240.6
64	5.0	115.9	225.7	321.5
128	22.2	478.3	960.1	1511.5

Table 2: Matrix Vector Multiply, Triangular Solve, Inner Product
Optimized
4K processors

We were able to prevent the need to move data when we followed the the matrix vector multiply by a sparse triangular solve because of the way in which we assigned data to processors. This method of data assignment did have the side effect of requiring us to perform the sparse matrix vector multiply in n consecutive phases. As one might expect, the use of this embedding can be seen to exact a performance penalty when compared to the embedding discussed in Section 3.1. For example we attained a computational speed of 64.2 MFlops for the problem with edge size 64. In Section 3.1, we attain a computational speed of only 27.1 MFlops with the embedding discussed in this section. The conforming inner product also had to be performed in n consecutive phases.

3.3. Comparison with Cray Timings

To put the CM-2 timings presented above into perspective we will present timings from a single head of a Cray X/MP for triangular solves and matrix vector multiplies arising from problems with the same structure as the problem we presented above. We used PCGPAK*, a commercially available Krylov space solver that handles general sparse matrices. In this program, the computation of the matrix vector multipiy and triangular solve in the iterative loop were vectorized using Cray Assembly Language (CAL). Computational speeds of 24 MFlops and 22 MFlops were obtained when performing matrix vector multiplies that arose from meshes with edge sizes 20 and 30 respectively. Note that the speed of the Cray did not increase with increasing problem size; on a vector machine such as the Cray X/MP, the computational rate depends chiefly on the problems structure rather than its size. By using a more specialized data structure that was well suited for vectorization, Ashcraft and Grimes [2] report speeds of 150 MFlops on a single head of an Cray X/MP for a matrix vector product. The speed of the matrix vector multiply in the largest problem described in Section 3.2 was 52.6 MFlops for the 4K processor machine. This speed should increase to 842 MFlops in a 64K processor machine on an appropriately scaled problem.

The triangular solve comparisons were also very favorable. For meshes with edge sizes of 20 and 30 a single head of an X/MP we achieved computational rates of 13 and 12 MFlops respectively using PCGPAK. Ashcraft and Grimes [2] report speeds of 25-40 MFlops on triangular solves in problems with the same structure; again their higher computational rates were achieved through the use of specialized data structures. The computational speed achieved by the largest problem described in Section 3.2 was 13.1 MFlops, which should increase to 210 MFlops in a full 64K machine for an appropriately scaled problem.

3.4. Summary of Performance on a Regular Three-Dimensional Mesh

The benchmarks described in this section imply that one can achieve respectable performance when one is able to make use of a highly tuned, special purpose PARIS program for carrying out the iterative portion of a large three-dimensional computation. Our experiences with the higher level languages *lisp and *C were less satisfactory. For instance, our *lisp versions of the sparse triangular solve described above required a factor of 20 more time to run than did the PARIS version.

3.5. Polynomial Preconditioning

The relatively slow speeds of the general router on the connection machine, as well as the relatively small parallelism in incomplete factorizations done in a fully "data-parallel"

*PCGPAK is a registered trademark of Scientific Computing Associates, Inc., New Haven, CT

Test Problem	GMRES (K)	Number of Iterations
SPE1	100	718
SPE2	100	> 2000
SPE3	100	> 2000
SPE4	100	120
SPE5	100	> 2000
5-pt	100	265
9-pt	100	140
7-pt	100	86
L9-pt	100	403
L7-pt	50	263

Table 3: Observed iteration counts for a selection of test problems using no preconditioning in the GMRES method.

way suggest that one may do better by using simpler preconditioners that vectorize well. An obvious choice for such a preconditioners is a polynomial preconditioner since it can be computed with highly parallel matrix vector products.

Because the particular choice of polynomial can have a significant effect on the effectiveness of the preconditioner, we looked instead at the performance of the GMRES method with no preconditioner but with a large number of "direction vectors." Roughly, the GMRES method minimizes the residual over combinations of $A^i x$ for $i = 0, \ldots, k-1$, and hence in some measure uses in d iterations the best possible polynomial of degree d. The test problems included the usual 5- and 9-point two-dimensional model problems, a 7-point three-dimensional model problem, and a set of "real-world" problems (the SPE set). By comparing results using GMRES with many direction vectors (large k) with GMRES with incomplete factorizations, we can estimate the number of additional iterations that a polynomial preconditioning would take (divide the number of iterations of GMRES without preconditioning by the degree of the polynomial to get an estimate of the number of iterations with polynomial preconditioning).

From our results, we can estimate the relative cost between using a highly parallel polynomial preconditioning and a less parallel but more effective incomplete factorization.

3.6. Brief Description of the Test Problems

In this section, we present the eight test problems used in our experiments.

Problem 1 (SPE1) This problem models the pressure equation in a sequential black oil simulation. The grid is $10 \times 10 \times 10$ with one unknown per gridpoint for a total of 1000 unknowns.

Problem 2 (SPE2) This problem arises from the thermal simulation of a steam injection process. The grid is $6 \times 6 \times 5$ with 6 unknowns per grid point giving 1080 unknowns. The matrix is a block seven point operator with 6×6 blocks.

Test Problem	GMRES (K)	Preconditioning	Number of iterations
SPE1	1	ILU(0)	70
SPE2	1	ILU(0)	23
SPE3	10	MILU(0)	51
SPE4	10	ILU(0)	39
SPE5	20	ILU(0)	104
5-pt	5	MILU(0)	44
9-pt	10	ILU(2)	47
7-pt	1	ILU(0)	39
L9-pt	10	ILU(1)	80
L7-pt	1	ILU(0)	31

Table 4: Observed iteration counts for a selection of test problems using incomplete factorization preconditioners in the GMRES method.

Problem 3 (SPE3) This problem comes from an IMPES simulation of a black oil model. The matrix is a seven point operator on a $35 \times 11 \times 13$ grid yielding 5005 equations.

Problem 4 (SPE4) This problem also comes from an IMPES simulation of a black oil model. The matrix is a seven point operator on a $16 \times 23 \times 3$ grid giving 1104 equations.

Problem 5 (SPE5) This problem arises from a fully-implicit, simultaneous solution simulation of a black oil model. It is a block seven point operator on a $16 \times 23 \times 3$ grid with 3×3 blocks yielding 3312 equations.

Problem 6 (5-Pt) This problem is a five point central difference discretization of the following equation on the unit square:

$$-\frac{\partial}{\partial x}(e^{-xy}\frac{\partial}{\partial x}u) - \frac{\partial}{\partial y}(e^{xy}\frac{\partial}{\partial y}u) + 2(x+y)(\frac{\partial}{\partial x}u + \frac{\partial}{\partial y}u) + (2 + \frac{1}{1+x+y})u = f$$

with Dirichlet boundary conditions and f chosen so that the exact solution is

$$u = x\,e^{xy}\sin(\pi x)\sin(\pi y).$$

The discretization grid is 63×63 giving 3969 unknowns. The L5-pt problem is the same problem with a 200×200 grid.

Problem 7 (9-pt) This problem is a nine point box scheme discretization for the following equation on the unit square:

$$-(\frac{\partial^2}{\partial x^2}u + \frac{\partial^2}{\partial y^2}u) + 2\frac{\partial}{\partial x}u + 2\frac{\partial}{\partial y}u = f$$

with Dirichlet boundary conditions and f chosen so that the exact solution is

$$u = x\,e^{xy}\sin(\pi x)\sin(\pi y).$$

The discretization grid is 63×63 giving 3969 equations. The L9-pt problem is the same problem with a 127×127 grid.

Problem 8 (7-pt) This problem is a seven point central difference discretization of the following equation on the unit cube:

$$-\frac{\partial}{\partial x}(e^{xy}\frac{\partial}{\partial x}u) - \frac{\partial}{\partial y}(e^{xy}\frac{\partial}{\partial y}u) - \frac{\partial}{\partial z}(e^{xy}\frac{\partial}{\partial z}u) + 80(x+y+z)\frac{\partial}{\partial x}u + (40 + \frac{1}{1+x+y+z})u = f$$

with Dirichlet boundary conditions and f chosen so that the exact solution is

$$u = (1-x)(1-y)(1-z)(1-e^{-x})(1-e^{-y})(1-e^{-z}).$$

The discretization grid is $20 \times 20 \times 20$ yielding 8000 equations. The L7-pt problem is the same problem with a $30 \times 30 \times 30$ grid.

3.7. Summary of Preconditioning Results

Table 3 shows the observed iteration counts for unpreconditioned GMRES with are large number of direction vectors. Table 4 shows the observed iteration counts for the same problems, using various flavors of incomplete factorization. In particular, the "real-world" SPE problems (except SPE4) take more than 10 times as many iterations using no preconditioning as when using incomplete factorizations. Very roughly, this means that using a tenth-degree polynomial as a preconditioner would give at best a similar iteration count (note that many of the SPE problems did not converge without preconditioning). Thus, an incomplete factorization that took as much time as such 10 matrix vector products would still be competitive and perhaps preferable (because of the better apparent behavior of incomplete factorizations on a wide range of problems).

It is also important to note that, in contrast, the simple model problems are solved fairly effectively by GMRES with no preconditioning, and hence polynomial preconditioning may be effective for these model problems. Still, we are interested in using a massively parallel machine for solving large and difficult linear systems, and, as long as the incomplete factorization solves can be done with reasonable efficiency, they are a better choice of preconditioner.

4. The Importance of Careful Embedding

In Section 3 we described how it was possible to achieve respectable rates of computation even on iterative loops that included a sparse triangular solve. In this section, we present some benchmarks that yield insight on what is required for achieving this high performance.

4.1. The Consequences of a Poor Mapping

We have found that 20 to 75 fold performance differences can be observed when we compared the performance of two versions of a sparse matrix vector multiply program (Table 5). The problem consisted of sweeps over sparse matrices generated by square domains of varying sizes with five point templates. The first version is explicitly mapped onto the machine in a way that allows us to utilize the CM-2's fast NEWS network for local communication. The other version uses a general router designed to carry out arbitrary patterns of interprocessor communication. Both versions were programmed in *Lisp and used the same data structures to represent the sparse matrices. For the 256 by 256 problem the timings for the explicitly mapped NEWS network code corresponded to a speed of 40.0 MFlops — the timings for the router version of the code achieved a speed of 0.5 MFlops. Note that the cost of computation does not vary significantly with the mapping and choice of communications method; this consistency provides a reassurance that the timing differences noted are actually due to communication related costs. The timings depicted here are averages obtained from 1000 iterations of the matrix vector multiply code. From this benchmark, it is quite clear that satisfactory performance cannot be obtained even from the most rudimentary sparse matrix code if one were to map sparse matrices onto the CM-2 without regard to data dependency patterns.

4.2. The Consequences of a Using the General Router in a Well-Mapped Problem

We next attempt to obtain some rough estimates of the performance we could expect if we were to write an iterative loop of a Krylov linear solver in a program capable of mapping reasonably general sparse matrices onto the CM-2. We must assume that the sparse matrix could arise from a mesh with irregularities. We consequently must use the general router rather than the NEWS network.

We will present benchmarks that attempt to quantify the effects of using the general router on a well-mapped problem. This should give us a best case estimate of the performance we might expect from using the general router. We examined the performance of versions of the PARIS program described in Section 3.2 in which we performed data *fetches* or data *sends* using the general router instead of performing data sends using the NEWS network. Note that in all of these cases, the problem was mapped so that only nearest neighbor communications were needed. The timings we obtained with data *fetches* carried out using the general router can be interpreted as best case estimates of what one could expect from a CM-2 executor that did not have access to a-priori information on dependency patterns. Table 6 depicts the average time per iteration required to solve a problem over a 64 by 64 by 64 grid. The time was averaged over 100 iterations. Note that in this case there is a roughly eight-fold performance difference between the optimized NEWS network program and the version employing the general router fetch instruction. As we mentioned in Section 3, on the 4K processor machine, the program employing the NEWS network achieves a speed of 27.14 MFlops on the matrix vector multiply and a

Grid Size	News Total (ms)	News Comp (ms)	General Total (ms)	General Comp. (ms)
64 x 64	2.25	0.93	45.56	0.99
128 x 128	4.62	1.88	189.44	1.89
256 x 256	13.11	5.46	1001.11	5.51

Table 5: Matrix Vector Multiply
Explicitly mapped News Net vs. General Router
4K processors

speed of 6.97 MFlops on the triangular solve. The fetch version of the general router program achieves a speed of 3.58 MFlops on the matrix vector multiply and 1.14 MFlops on the triangular solve.

In Table 7 we compare timings obtained through the use of the NEWS network and the fetch instruction from the general router for three-dimensional meshes with varying edge size. We also present the ratio of the fetch timings to the NEWS net timings. For both the matrix vector multiply and the triangular solve, the ratio between the NEWS net and router timings remain roughly constant for all meshes with edge size up to 64. The router timings become relatively less efficient for the mesh with edge size 128. When we solve this problem with a 128 edge sized mesh, 4 virtual processors are assigned to each actual processor. As described above, the assignment of multiple virtual processors to each actual processor can have the effect of reducing communications overhead; less information needs to be exchanged between processors. When the router is used, each virtual processor must go through the overhead of using the router, even when only sending information to another virtual processor assigned to the same actual processor.

Function	NEWS (ms)	Router Send (ms)	Router Fetch (ms)
Matrix Vector	116	249	876
Solve	226	374	1376
Total	342	606	2247

Table 6: Three-Dimensional Mesh Sweep and Solve
News Net vs. Send and Fetch General Router
4K processors

5. Conclusions

In Section 3, we presented what we might regard as best case timings for the sparse matrix vector multiplies, sparse triangular solves, and inner products that constitute the iterative portion of Krylov space linear solvers when the solvers use incompletely factored matrices for preconditioning. We performed timings for large three-dimensional model problems over cube-shaped domains discretized with a seven point template. The highest computational rate we were able to achieve for the sparse triangular solve was 13.1 MFlops on 4K processors, which would correspond to a timing of 210 MFlops on an appropriately scaled problem on a 64K processor machine. The highest computational speed we achieved for a matrix vector multiply was 64.2 MFlops, which would correspond to to a speed of 1027.0 MFlops in a 64K processor machine. Thus for regularly structured problems, the CM-2 achieves impressive computational speeds. For all three compute-intensive kernels the projected speeds on a full 64K processor machine exceed the speeds that can be achieved by carefully vectorized code on a single head of a Cray X/MP by a factor of at least 5 to 6. These timings indicate that we can build a preconditioned Krylov space solver which is both close to optimal with respect to the required number of floating point operations and yet performs at a rate of computation far exceeding that of a highly optimized version running on a single-headed Cray-X/MP.

The timings on the CM-2 were highly dependent on the use of our specialized programs. These programs mapped the problem domain onto the processor topology very

Edge Size	MVM NEWS (ms)	MVM Router (ms)	MVM Ratio	Solve NEWS (ms)	Solve Router (ms)	Solve Ratio
16	29	208	7.2	52	328	6.3
32	57	441	7.7	107	664	6.2
64	116	876	7.6	226	1376	6.1
128	478	4581	9.6	961	7098	7.3

Table 7: Three-Dimensional Mesh Sweep and Solve
News Net vs. General Router—Varying Mesh Size 4K processors

carefully and used the optimized local NEWS communications network. We explored the consequences of relaxing these restrictions in a variety of ways. Performance degraded very significantly as one abandoned the optimized NEWS communications network and abandoned the careful mapping of a problem domain to the CM-2 processor topology. In some cases we noted 20 to 75 fold degradations in performance as these constraints were relaxed.

The results of our benchmarks clearly demonstrate that one can obtain extremely high performance on Krylov space linear solvers on the CM-2. It appears that it would be quite feasible to construct a preconditioned Krylov solver capable of solving systems arising from partial differential equations discretized using one of a fixed set of finite difference templates and obtain extraordinary performance on the CM-2. Taking into account the relative performance of such a solver on the CM-2 and the Cray-X/MP and the relative prices of these computer systems, an appropriate preconditioned Krylov solver on the CM-2 would revolutionize the computational solution of engineering and scientific problems involving three-dimensional simulations.

References

[1] E. Anderson, *Solving Sparse Triangular Linear Systems on Parallel Computers*, Report 794, UIUC, June 1988.

[2] C. Ashcraft and Roger G. Grimes, *On Vectorizing Incomplete Factorization and SSOR Preconditioners*, Technical Report ETA-TR-41, Boeing Computer Services, December 1986.

[3] D. Baxter, J. Saltz, M. Schultz, S. Eisentstat and K. Crowley, An Experimental Study of Methods for Parallel Preconditioned Krylov Methods, *Proceedings of the 1988 Hypercube Multiprocessor Conference, Pasadena CA*, January 1988.

[4] D. Baxter and J. Saltz and M. Schultz and S. Eisenstat, Preconditioned Krylov Solvers and Methods for Runtime Loop Parallelization, *Proceedings of the 4th International Supercomputing Conf., St. Clara, CA*, April 1989.

[5] H. Berryman and J. Saltz and W. Gropp, *Krylov Methods with Incomplete Factorization Preconditioners on the CM-2*, Submitted to Journal of Parallel and Distributed Computing, Feb (1989).

[6] H. Berryman and W. Gropp and J. Saltz, Krylov Methods and the CM-2, *Proceedings of the 4th International Supercomputing Conf., St. Clara, CA*, April 1989.

[7] Todd Dupont, Richard P. Kendall and H. H. Rachford Jr., *An approximate factorization procedure for solving self-adjoint elliptic difference equations*, SIAM Journal on Numerical Analysis, 5 (1968), pp. 559-573.

[8] H. Elman, *Personal Communication*, with the authors, University of Maryland, 1988.

[9] J. Saltz, *Aggregation Methods for Solving Sparse Triangular Systems on Multiprocessors*, SIAM J. Sci. and Stat. Computation., to appear Sept (1989).

[10] J. A. Meijerink and H. A. van der Vorst, *Guidelines for the usage of incomplete decompositions in solving sets of linear equations as occur in practical problems*, Journal of Computational Physics, 44 (1981), pp. 134-155.

[11] ———, *An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix*, Mathematics of Computation, 31 (1977), pp. 148-162.

[12] Herbert L. Stone, *Iterative solution of implicit approximations of multidimensional partial differential equatins*, SIAM Journal on Numerical Analysis, 5 (1968), pp. 530-558.

[13] T. F. Chan and C. C. Kuo and C. Tong, *Parallel Elliptic Preconditioners: Fourier Analysis and Performance on the Connection Machine*, Report CAM 88-22, UCLA, August 1988.

[14] Y. Saad, M. Schultz, *Parallel Implementations of Preconditioned Conjugate Gradient Methods*, Department of Computer Science YALEU/DCS/TR-425, Yale University, October 1985.